## *Amendments to the Specification*

Kindly amend paragraph [0005] as follows:

[0005] CORBA, ~~COBRA,~~ another interface description language, is also capable of describing only interfaces. The language lacks the ability to describe the actual structure of objects and blocks. Furthermore, CORBA is ~~COBRA is~~ not extensible. Thus, it only allows developers to specify a fixed set of additional information.

Kindly delete the Table of Contents section in its entirety on page 5, lines 2-38 after paragraph [0017].

Kindly amend paragraph [0070] as follows:

[0070] FIG. 1 is a diagram of an example embodiment of system 100, a system in which the ODL of the present invention is utilized. Game developer 110 writes C++ code 135 and ODL code 115. In the embodiment depicted in FIG. 1, ODL code 115 and C++ code 135 may be in separate files. ODL code 115 describes the structure and elements of the computer program written by developer 110. After the developer writes ODL code 115, code 115 is processed by ~~IGEN~~ODL compiler 120. ~~IGEN~~ODL compiler 120 produces code 125, which corresponds to ODL Code 115. In one embodiment, code 125 may be machine code. In another embodiment, Code 125 may be C++ human-readable code. Code 125 then passes to C++ compiler 130.

Kindly amend paragraph [0072] as follows:

Referring to FIG. 2, an alternative embodiment of the present invention, C++ declarations 210a and ODL code 115 are developed together in a single file. In such a case, ODL code 115 and C++ declarations 210a must be separated, as shown in FIG. 2. Header file 210 is written by game developer 110 and contains C++ declarations 210a and ODL code 115. Header file 210 is passed to ~~IGEN~~ module 215. More specifically, header file 210 is passed to separator 220, located within ~~IGEN~~ module 215. In one

embodiment, separator 220, combiner 240, and ~~IGEN~~ODL compiler 120 may be contained in their own separate modules.

Kindly amend paragraph [0073] as follows:

Referring to the example embodiment in FIG. 2, Separator 220 extracts a copy of ODL code 115 from header file 210 written by the developer. Separator 220 then passes a copy of ODL code 115 to ~~IGEN~~ODL compiler 120. ~~IGEN~~ODL compiler 120 compiles ODL code 115 and generates code 125. In another embodiment ~~IGEN~~ODL compiler 120 is actually a translator and translates ODL code 115 into human-readable C++ source code. After the compilation, ~~IGEN~~ODL compiler 120 passes code 125 to combiner 240. Separator 220 passes the C++ declarations 210a (also represented in FIG. 2 by C++ User Preamble 222, C++ User Preobject 223, C++ User Members 224, C++ User Post-object 225, and C++ User Postamble 226) to Combiner 240.

Kindly amend paragraph [0080] as follows:

Referring to FIG. 4, a flowchart is shown representing the general operational flow, according to an embodiment of the system in which the present invention is utilized. More specifically, flowchart 400 depicts an example routine for processing header file 210. Routine 400 begins with step 402. In step 402, header file 210, containing C++ declarations 210a and ODL code 115, is read by ~~IGEN~~ module 215. In step 404, separator 220 copies ODL code 115 from header file 210 and passes the copied ODL code 115 to ~~IGEN~~ODL compiler 120. In step 406, ODL code 115 is compiled to obtain code 125. In step 408, ODL code 125 is combined with C++ code 135 (222-226 in FIG. 2) and ODL code 115, forming a new header file.

Kindly amend paragraph [0082] as follows:

Corresponding C++ code 520 is the result of ~~IGEN~~ compilation of ODL code 510. Corresponding C++ code 520 contains a class named "Dude" with public access, and the variables x and y, just as ODL code 510 contains.